

Solving Classic AI Problems with Search Algorithm

1. Introduction

Artificial Intelligence (AI) search algorithms are fundamental techniques used to solve problems that involve finding a sequence of actions leading from an initial state to a goal state. This project aims to provide hands-on experience in implementing and analyzing classic search algorithms by applying them to a well-known AI problem.

In this project, we focus on a **Robot Pathfinding Problem in a Grid with Obstacles**. The robot must navigate from a starting position to a goal position while avoiding obstacles. Multiple search strategies are implemented and compared to understand their performance, efficiency, and suitability for different scenarios.

2. Why This Problem?

The **Robot Pathfinding Problem** was chosen because it is a **classic AI problem** that clearly demonstrates the behavior and efficiency of different search algorithms. It is suitable for this project for several reasons:

1. **Clear Goal and Constraints:** The robot must move from a start position to a goal while avoiding obstacles. This makes the problem easy to model and visualize.
2. **State Space Exploration:** The problem requires exploring multiple paths in a grid, which allows us to compare algorithms based on nodes explored, memory usage, and execution time.

3. **Applicability of Multiple Algorithms:** Both uninformed (BFS, DFS, UCS, IDS) and informed (A*, Genetic Algorithm) search strategies can be applied, providing a meaningful comparison.
 4. **Measurable Performance Metrics:** The problem allows measurement of **path cost, nodes explored, space usage, and execution time**, which are important for evaluating AI algorithms.
 5. **Real-World Relevance:** Pathfinding is widely used in robotics, gaming, and navigation systems, making this problem practical as well as educational
-

2. Problem Definition

2.1 Problem Description

The environment is represented as a two-dimensional grid. Each cell in the grid can be:

- Free space (0)
- Obstacle (1)
- Start position (S)
- Goal position (G)

The robot can move one step at a time in four directions: up, down, left, and right. Each move has a uniform cost of 1.

2.2 State Representation

A state is represented as the robot's current position in the grid:

State = (x, y)

2.3 Initial State

The initial state is the cell marked with S.

2.4 Goal State

The goal state is the cell marked with G.

2.5 Actions

- Move Up
- Move Down
- Move Left
- Move Right

2.6 Path Cost

Each movement has a cost of 1. The total path cost is the number of steps taken from the start to the goal.

3. Grid Environment

The grid used in this project is shown below:

S	0	0	1	0
1	1	0	1	0
0	0	0	0	0
0	1	1	1	0
0	0	0	G	0

4. Implemented Search Algorithms

4.1 Breadth-First Search (BFS)

BFS explores the search space level by level. It guarantees finding the shortest path when all step costs are equal.

Characteristics:

- Complete and optimal
- High memory consumption

4.2 Depth-First Search (DFS)

DFS explores one branch of the search tree as deeply as possible before backtracking.

Characteristics:

- Low memory usage
- Not optimal
- May get trapped in deep paths

4.3 Uniform Cost Search (UCS)

UCS expands the node with the lowest path cost. It is optimal when all costs are non-negative.

Characteristics:

- Complete and optimal
- Higher time and space complexity

4.4 Iterative Deepening Search (IDS)

IDS combines the advantages of BFS and DFS by gradually increasing the depth limit.

Characteristics:

- Complete and optimal
- Lower memory usage than BFS
- Repeated computations

4.5 A* Search

A* is an informed search algorithm that uses a heuristic function to guide the search toward the goal.

Heuristic Function: Manhattan Distance:

$$h(n) = |x_1 - x_2| + |y_1 - y_2|$$

This heuristic is admissible and consistent, ensuring optimal solutions.

Characteristics:

- Complete and optimal
- Fast and efficient

4.7 Genetic Algorithm

The Genetic Algorithm is an optimization technique inspired by natural selection.

Representation:

- Chromosome: A possible path
- Fitness Function: Inverse of path length

Operators:

- Selection
- Crossover
- Mutation

Characteristics:

- Good for optimization
- Does not guarantee optimal solutions

5. Performance Analysis

The algorithms were evaluated based on:

- Time complexity
- Space complexity
- Optimality
- Path cost

6. Comparison of Algorithms

Algorithm	Path Cost	Nodes Explored	Space Usage	Execution Time (ms)	Optimal
BFS	9	16	18	0.087	Yes
DFS	9	10	20	0.045	Yes
UCS	9	17	17	0.07	Yes
IDS	9	68	20	0.151	Yes
A*	9	11	12	0.098	Yes
Genetic	1 (failed)	1000	10	1.813	No

7. Results and Discussion

- **Efficiency:** A* outperforms other algorithms in terms of nodes explored and memory usage.
 - **Optimality:** BFS, UCS, IDS, DFS, and A* guarantee optimal paths; Genetic Algorithm may fail.
 - **Trade-offs:**
 - BFS: High memory usage but guaranteed shortest path.
 - DFS: Fast but not memory-efficient; may fail in larger grids.
 - UCS: Optimal but slightly slower than BFS.
 - IDS: Memory-efficient but explores more nodes.
 - Genetic: Good for optimization, but unreliable for exact pathfinding
-

8. Conclusion

- For small grids, **all classical search algorithms** except Genetic Algorithm can find the optimal path efficiently.
 - A* provides the best balance between **efficiency, memory usage, and optimality**.
 - Genetic Algorithm is better suited for optimization problems, not strict pathfinding
-

9. References

- Russell, S. & Norvig, P. *Artificial Intelligence: A Modern Approach*
- Course Lecture Notes

